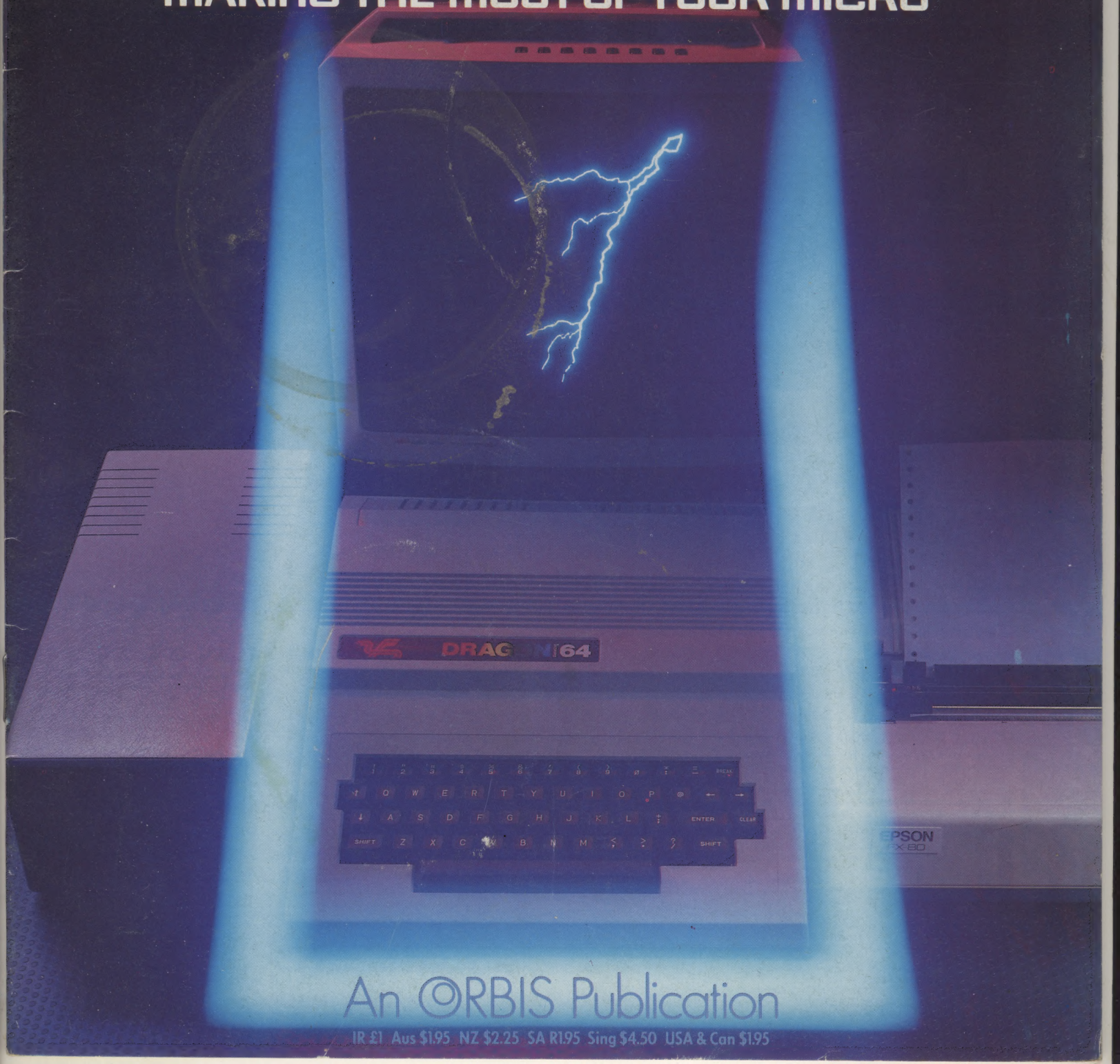
ISSN 0265-2919

80p

10

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

SMOOTH OPERATORS Computers are provided with a built-in program that operates their vital housekeeping routines 181

HARDWARE

ATARI UPDATE We review the new XL range of Atari computers 189

SOFTWARE

MEMORY MANAGERS Introducing a new series on how the computer creates and manages files 184

LINES OF ENQUIRY Continuing our analysis of how business software can handle company stock control 192

COMPUTER SCIENCE

ON THE LEVEL We design our own error checking circuit 186

JARGON

FROM BIT TO BREADBOARD A weekly glossary of computing terms 188

MACHINE CODE

MODES OF ADDRESS We explain addressing modes — an essential feature of machine code programming 196

PROFILE

BRANCHING OUT The Tandy Corporation has developed from a small electrical retailer to a multinational computer company 199

WORKSHOP

TESTING TIME We pause to recap before tackling more advanced projects 194

Next Week

- **Sharp's PC-5000 uses the very latest technology to bring you one of the most powerful portables ever. We look at a machine that has brought desktop computing capacity to the portable world.**
- **In the first of a series on graphics for all the popular micros, we look at a submarine hunter game for the Commodore 64.**
- **Video discs and laser discs combined with home computers promise to make large quantities of information more accessible in the home than ever before.**

COMING VERY SOON

FREE

MACHINE CODE MONITOR AND ASSEMBLER PROGRAM ON CASSETTE

Written specially for Home Computer Advanced Course readers

COVER PHOTOGRAPHY BY PAUL CHAVE

Editor Max Phillips; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Brian Morris; **Picture Editor** Claudia Zeff; **Designers** Hazel Bennington, Julian Dorr; **Sub Editor** Robert Pickering; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Lisa Kelly, Steven Colwill, Geoff Bains, Tony Harrington, Richard Pawson, Mike Wesley; **Consultant Editor** Gareth Jefferson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax. MALTA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE &

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

SMOOTH OPERATORS

PAUL CHAVE

The Manager
Every computer has some form of operating system — a program that manages the running of the computer and controls all the devices attached to the system

The operating system is a vital part of any computer because it forms a link between the hardware and software. Yet because operating systems do most of their work in the background, many people, especially home computer users, are hardly aware they exist. To help put the record straight we present an overview of operating systems.

High-level languages allow the programmer to be isolated from the inner workings of the CPU and make for greater portability of programs between one system and another. Provided that a language is reasonably standardised, instructions should work on any machine supporting the language. The interpreter or compiler that processes the high-level language source code takes care of the details of memory allocation and so on. The high-level interpreter or compiler is also a program and it must be loaded into main memory before it can convert the high-level source code into object code instructions ready for execution. BASICs in ROM are already, permanently, present in memory and ready for use as soon as the machine is switched on.

However, there's more to operating a computer than just having a program in it able to convert source code into machine code. There needs to be yet another program running in the background that concerns itself with the 'housekeeping'. As an example of housekeeping, consider the problem of getting a letter typed in on the keyboard to appear on the screen. Somewhere in memory there has to be a program telling the CPU constantly to check the keyboard to see if a key has been pressed. If one has, the program has to work out which key it was, and then it has to instruct the video circuitry to produce the right pattern of dots, in the right sequence, for output to the screen. Activities such as this are said to be 'transparent' to the user.

Similarly, when a command such as CSAVE is issued to save a file on cassette, the programmer is not concerned about how the data is converted into a form suitable for cassette storage; it's all part of the operating system.

The operating system is the background program that runs continuously, supervising everything else. A slight source of confusion arises when the computer in question is a ROM-based small computer system with a built-in BASIC, because the BASIC and the operating system are often held on the same ROM. In its simplest

form, then, an internal ROM will contain all the software neded to run the system, apart from applications programs (games, word processors etc.) loaded in or written by the user. Part of this ROM will contain the code needed for converting applications programs written in BASIC into machine code (the *interpreter*); part will contain the code needed for entering and modifying user-written programs (the *editor*); and part will contain the housekeeping software needed for looking after the keyboard, displaying characters and graphics, accepting data from cassettes and allocating it to the right parts of memory and so on (the *monitor*).

The term 'monitor', not to be confused with a television or display monitor, is roughly synonymous with 'operating system'. In its simplest form, the monitor is able to do little more than accept instructions in machine code, place them in the right memory location, and supervise their execution. Once the housekeeping becomes a little more advanced than this, the monitor itself tends to be referred to as the operating system.

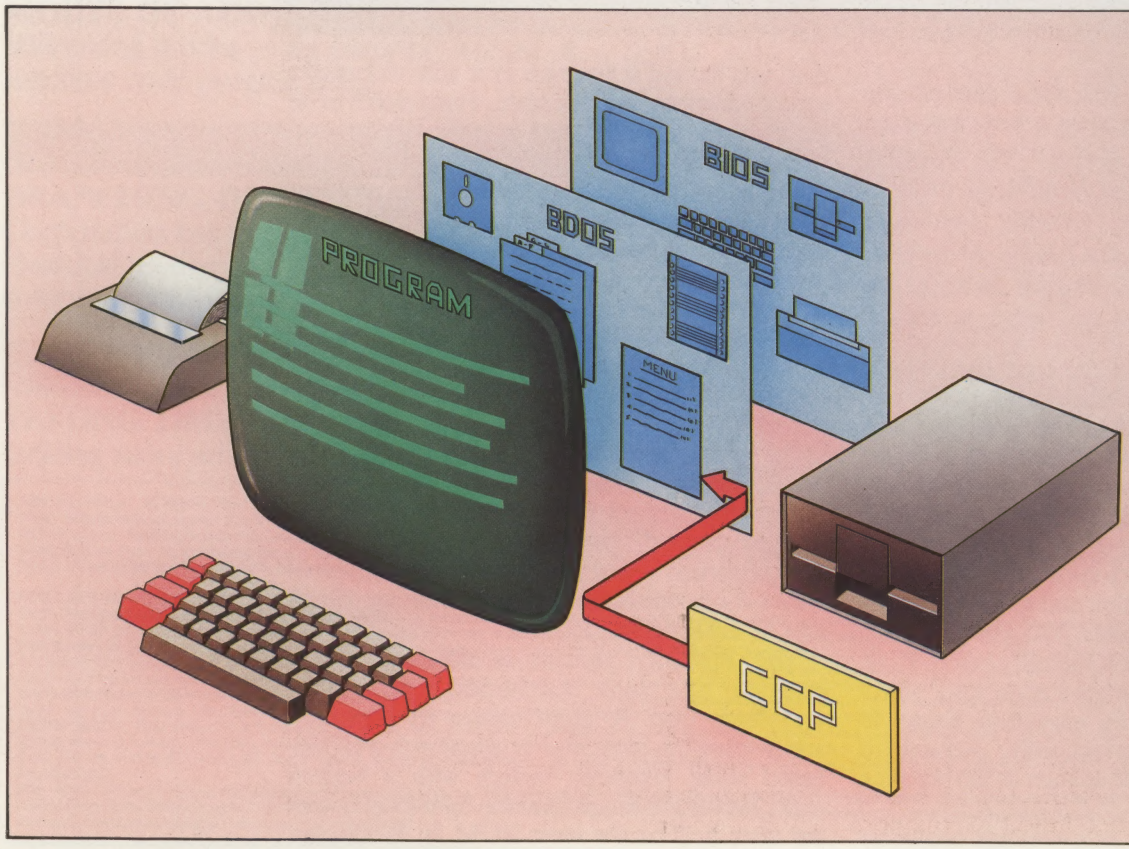
At the other extreme are the disk-based computers, often used in offices as small business systems, which have powerful operating systems. Before considering intermediate computers such as the Apple, we will consider the kind of operating system needed by a disk-only system.

A computer system entirely based on floppy disks for its software will normally have very little stored permanently in ROM apart from a bootstrap loader (see page 188) and a few housekeeping routines. When such a computer is switched on, the bootstrap loader contains just enough machine code to instruct the CPU how to access a disk drive and load the operating system into main memory.

The operating system loaded into RAM has to be able to do more than the operating systems in more conventional ROM-based systems. It becomes what is called a DOS or disk operating system. An operating system such as this extends normal housekeeping functions by adding commands that act directly on the files stored on the disk. The kind of commands expected from a disk operating system include commands to list the names of the files stored on the disk, to erase or re-name files, and to copy files from the disk into main memory or onto other disks.

The operating systems of simpler, ROM-based systems usually do not have such sophisticated file-handling commands, and may have nothing more than a simple command to load a named file from tape or to store a file under a given name on a tape. A sophisticated operating system will know the exact address location on disk or tape storage where any file is stored. Less advanced operating systems may be able to do nothing more than search through all the files present until the one named is encountered, and then load it: or to write a file under a given file name on the tape at whatever point the tape happens to be when the command is issued.

Computer systems intermediate between the two are well exemplified by the BBC Micro, which is equally adept at handling cassette or disk files, using largely the same commands. The operating system resides in a ROM, but it is a physically separate ROM and can be thought of



Operation Controls

CP/M is constructed in layers to enable both itself and the programs written for it to be moved easily from one micro to another. The user's program works through the BDOS (Basic Disk Operating System), a set collection of routines that provide standard functions and facilities for the program. The BDOS is the same in all machines, so a program that uses it will work on any CP/M system. When the BDOS needs to perform a particular task, it calls the BIOS (Basic Input/Output System). This is specially adapted for every machine and is the software link between the BDOS and the physical components of the computer.

CP/M also includes the CCP (Command Control Program), which allows the user to enter commands to run programs, rename and delete files and generally control the computer system

KEVIN JONES

as an operating system that has been loaded into memory from an external memory device. Its functions include file-handling commands considerably more advanced than those encountered on other ROM-only computers such as the Spectrum.

The operating system used by a computer, then, can be seen as a program with the function of sitting between the user and the rest of the computer system, including its CPU, systems software (such as programming languages) and applications software.

PORTABILITY

The ability to use software on more than one computer system is known as *portability*. There are, broadly, two aspects to this. The first is the fact that different processors require different instruction sets in order to perform equivalent operations. Thus, machine code instructions to, say, add together the contents of two memory locations would have one form if written for the 6502 (used in the Apple) and an entirely different form if the same operation were required on a Z80 computer such as the Spectrum. The problem of converting high-level code into suitable machine code is, however, the responsibility of the interpreter or compiler used. Different interpreters and compilers have to be written for each different CPU.

There is, however, a separate problem affecting software portability. Even when the same CPU is used, as in the Apple and the BBC, there are other complications. Different address locations are used for the video memory, different codes are needed to move the cursor about the screen, different input and output facilities are provided, and so on.

To overcome this problem, generic disk operating systems were developed that would allow all software written for, say, one disk-based Z80 computer to run on any other disk-based Z80 computer having the same operating system. The best known of these disk operating systems is CP/M (Control Program/Microcomputers).

Disk operating systems such as CP/M are essentially a development from the more machine-specific monitors and operating systems, but they represent a major advance in terms of software portability. Any program written to run under a generic operating system such as CP/M or MS-DOS will run on any computer with that operating system, provided the software does not try to make use of any special features (such as sound effects) specific to one machine. The operating system software itself is supplied in standard form by its developers to the computer manufacturer. All the hardware manufacturer has to do is to rewrite a small machine-dependent portion of the program.

Disk operating systems vary considerably in their complexity and capabilities, but the simpler ones such as CP/M and MS-DOS comprise essentially three parts: the command processor, the basic disk operating system (BDOS) and the basic input/output system (BIOS). Of these parts, the only one relevant to portability is the BIOS. The BIOS is a separate part of the program that contains all the routines needed to handle the peripherals, including the screen and the keyboard, and it has to be specially configured for each new computer design. Any program that runs under the control of this operating system will interface with the computer through the BIOS. The BIOS, then, will handle such things as getting characters fom the keyboard, outputting characters to the screen or printer, addressing the disks, and so on.

The BDOS consists of the parts of the operating system that are not device-specific (i.e. generalised routines for handling the screen, printer, disk drives etc.). These parts of the program do not need to change between implementations. The BDOS and BIOS together correspond roughly to the monitor or operating system found in ROM-based computers.

The command processor is the part of the program that handles operating system commands typed in from the keyboard. Typical commands include those that load files from disk into main memory, list the file names present on the disk and erase or rename files on the disk.

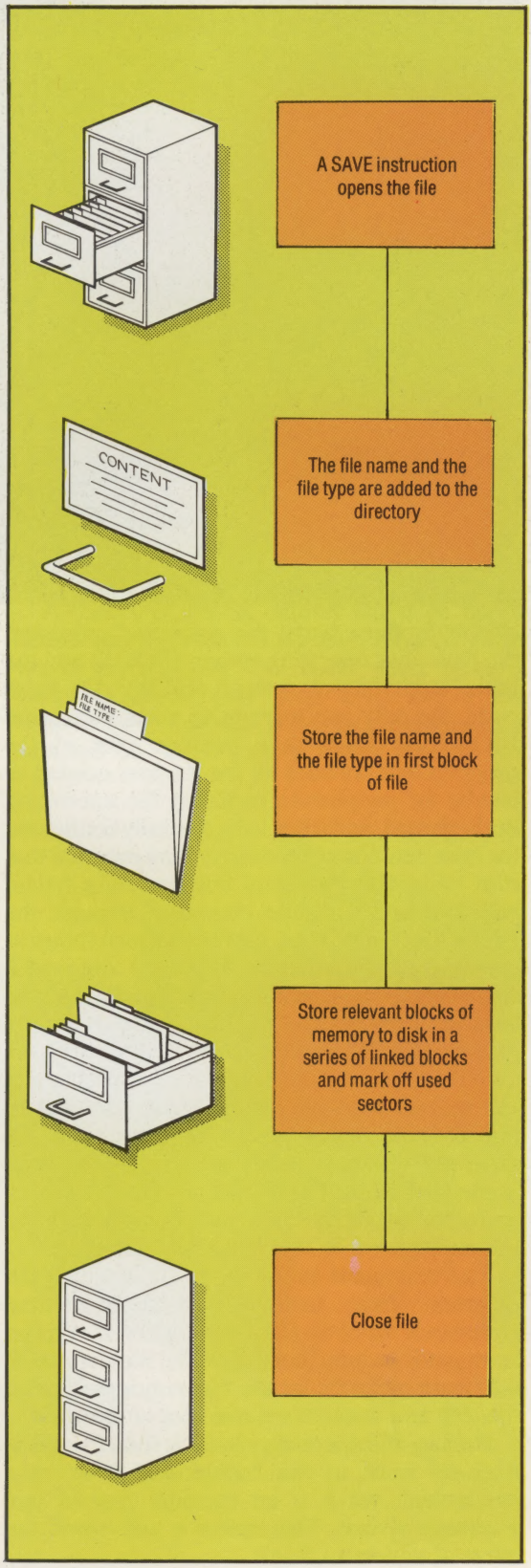
Because the operating system is something that does its work in the background it is often overlooked, yet it is an essential part of any computer system. This makes it well worth the effort of understanding it.

IAN McKINNELL

Key To Success
The Osborne 1 owes much of its success to being a CP/M-based computer. It comes with some of the most popular CP/M programs, including WordStar, SuperCalc and MBASIC

MEMORY MANAGERS

Saving A Binary File
A binary file is simply a copy of a portion of memory. This could consist of a program in memory, a screen image and so on. The filing process is straightforward; after an entry for the file has been made in the directory, the data is written as a linked series of sectors. The DOS will also keep a list of the sectors used, so that they are not overwritten when another file is created


A SAVE instruction opens the file
CONTENT
The file name and the file type are added to the directory
FILE NAME:
FILE TYPE:
Store the file name and the file type in first block of file
Store relevant blocks of memory to disk in a series of linked blocks and mark off used sectors
Close file

We have already discussed the advantages of disk storage systems over tape systems (see page 4) and looked at the disk systems used by the more popular home computers. The series of articles we begin here examines the standard methods of file handling used by disk storage systems: binary, sequential and random access files.

File is a very apt word when used in reference to computer storage, since direct analogies can be drawn with the method of storing documents and records in a filing cabinet system. Bearing this in mind, we will first discuss why filing systems are necessary in home computers.

In order to efficiently 'manage' our day-to-day lives, it is necessary to maintain the most accurate record of our experiences, monetary transactions, social appointments and so on. Most people use a diary/address book and keep tabs on a bank account by filling out the cheque stubs. This need to store information and, more importantly, recall it simply and easily is amplified many times when dealing with the large amount of constantly changing information inherent in any business or project involving different people, places, objects and circumstances. As such enterprises grow, the management of information becomes increasingly complex, and most problems experienced by ventures of this type can be attributed to mismanagement and misinterpretation of information. A simple and efficient method of storing, indexing and retrieving data is the essence of good management. Capable administrators understand the need for good filing techniques, structuring their methods of storage according to the type, volume and rate-of-change of the information under their control.

These principles remain the same when applied to the fundamental ability of computer systems to manipulate and accurately store vast quantities of information at an incredibly high speed. At the centre of such a system is the computer's 'administrator' — the disk operating system (DOS) — which works perfectly well, provided it is given the correct information and asked the right questions by the 'manager' (i.e. you or your program). So, computer storage systems are only as efficient as the data structure (or filing system) adopted by the DOS, and the way in which the DOS is used.

The standard methods of file handling used by microcomputer systems were originally developed for mainframe and mini computers. They can be broken down into three systems:

Disk Directory

On the screen is a disk directory produced by the CP/M utility STAT, which gives considerably more information than most directory displays

Recs
The number of records in the file can vary; here the records are 128 bytes long.

Bytes
The length of the file in Kbytes

Ext
The extent is an alternative measure of the disk space occupied by the file

Acc
Access: a file can be marked for reading and writing (R/W), or for reading only (R/O)

File Name
The full file name starts with a drive name (A: or B:), followed by the file name (AUTOST, for example), followed by the file extension (.COM, for example), which may give some information about the file's contents

```
 Recs   Bytes   Ext  Acc  Filename
    0      0K     1  R/W  B:ACNTLIST.DTA
   11      2K     1  R/W  B:ANT
   10      2K     1  R/W  B:ANT.BAK
   64      8K     1  R/W  B:ASM.COM
   16      2K     1  R/O  B:CODELIST.DTA
   16      2K     1  R/W  B:AUTOST.COM
    1      1K     1  R/W  B:COMPANY.DTA
    2      1K     1  R/W  B:CONTROL.DTA
   34      5K     1  R/W  B:COPY.COM
   40      5K     1  R/W  B:DDT.COM
    4      1K     1  R/W  B:DUMP.COM
  250     32K     2  R/W  B:INSTALL.COM
    4      1K     1  R/W  B:JUNK
   16      2K     1  R/W  B:LOAD.COM
    6      1K     1  R/W  B:MICROLIN
   40      5K     1  R/W  B:ML.COM
   86     11K     1  R/W  B:MOVCPM.COM
   58      8K     1  R/W  B:PIP.COM
    2      1K     1  R/W  B:SCREEN.ASM
    1      1K     1  R/W  B:SCREEN.COM
    4      1K     1  R/W  B:SCREEN.DOC
   42      6K     1  R/W  B:STAT.COM
Bytes Remaining On B: 85K
```

binary, sequential, and random access files. We will look at each of these methods separately.

BINARY FILES

A binary file is simply a copy of a portion of user memory, a good example being a SAVEd program. Imagine the area of RAM available to the user as a simple notepad. If you were keen to preserve vital notes or interesting drawings, you would tear the relevant pages off and keep them where they were handy. Binary files work in the same way. When a SAVE command is given, the DOS stores the FILENAME on disk, marking it in some special way as a binary file and then copies the relevant area of memory byte-by-byte to the disk. The program is stored in linked blocks (with the markers at the end of each block indicating where the next block begins) until the end of the program or data is reached. The last block finishes with an end-of-file marker. Using our analogy we could say that we have named and stored a page from our notepad in a filing cabinet drawer, and added the name of the file to the contents list on the drawer.

On being RETURNed or ENTERed, a BASIC program line is compressed into a tokenised form in which the BASIC keywords are coded by the BASIC interpreter into one-byte numbers. These can be manipulated more easily and decoded back into text for LISTing purposes. As a binary file is a RAM image it can also store ASCII codes and binary data. The facility to save ASCII files is useful for storing the contents of screen memory, for example, so that screen displays can be SAVEd and later LOADed back into the same area of memory easily. In addition, some disk operating systems and BASICs allow a BASIC program to be stored in ASCII form. This enables the untokenised program to be edited as a text file in machines with sophisticated editor programs.

Binary files are very simple to use and manage, but they are limited by two factors. First of all, it is only possible to SAVE the relevant information as one continuous section of data. As a consequence, the information must be retrieved in the same manner and therefore binary files must be LOADed back into memory in their entirety. Secondly, the maximum size of a file is limited by the amount of RAM available to the user.

In the next instalment of the course we will look at sequential files, which allow a file to be as long as is required (within the limits of disk space), and random access files, which use different methods to allow the DOS to store data in such a way that it can be freely retrieved and updated.

ON THE LEVEL

We are now at a stage in the Logic course where we can design quite complex computer circuits. In this instalment, we will follow through the whole design process — from initial specification, through truth table and simplified Boolean expression, to finished circuit diagram — for a parity bit generating circuit and a priority encoder.

Before beginning to look at the design of these two advanced applications, we will first take a detailed look at another important logic gate — the Exclusive OR (XOR) gate. This gate has already been briefly considered (see page 47), but we have not yet given the Boolean algebra or circuit diagram symbols for it:

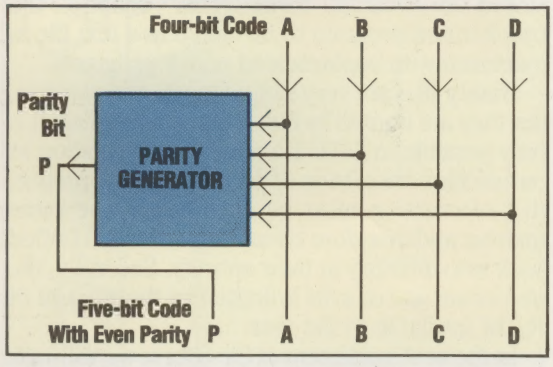
Truth Table			Circuit Symbol
A	B	C	
0	0	0	A, B → XOR → C
0	1	1	
1	0	1	
1	1	0	Boolean Symbol = ⊕

From the truth table, it can be seen that the output C can be expressed in two ways:

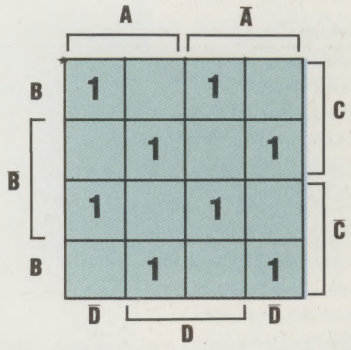
a) $C = A \oplus B = \overline{A}.B + A.\overline{B}$
b) $\overline{C} = \overline{A \oplus B} = \overline{A}.\overline{B} + A.B$

The second expression is formed by considering the cases when C is not one (i.e. zero). This gate will be of particular use in our first application.

A PARITY BIT GENERATOR



Parity is an important concept in the design of data transmission systems. The parity bit (see previous diagram) of a binary code is added to the rest of the code in order to make all the codes transmitted have an even number of ones. (Another convention is to make all the codes contain an odd number of ones — this is known as *odd parity*). A parity bit acts as a checking system to ensure that the correct transmission has taken place. The circuit we shall design will accept a four-bit code and produce an appropriate parity bit. With a small modification the circuit may also act as a parity checker of incoming data. The truth table for this circuit is given in the margin. Representing these values on a k-map gives:



The symmetrical pattern produced on the k-map, unfortunately, does not allow simplification because no groups can be formed. The resulting expression for P is:

$$P=\overline{A}.\overline{B}.\overline{C}.D+\overline{A}.\overline{B}.C.\overline{D}+\overline{A}.B.\overline{C}.\overline{D}+\overline{A}.B.C.D$$
$$+A.\overline{B}.\overline{C}.\overline{D}+A.\overline{B}.C.D+A.B.\overline{C}.D+A.B.C.\overline{D}$$

By grouping the red terms together and the blue terms together, we can simplify the expression:

$$P=(\overline{A}.\overline{B}+A.B).(\overline{C}.D+C.\overline{D})$$
$$+(A.\overline{B}+\overline{A}.B).(\overline{C}.\overline{D}+C.D)$$

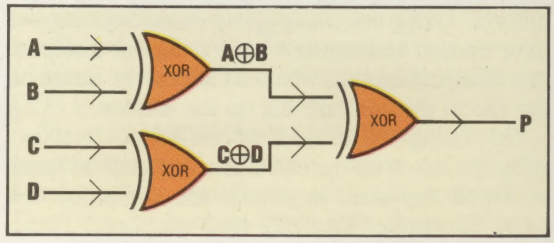
Now, by referring to the expressions for an XOR gate that we introduced at the start of this article, we can further simplify the expression to get:

$$P=(\overline{A\oplus B}).(C\oplus D)+(A\oplus B).(\overline{C\oplus D})$$

By considering each bracketed term as an input to an XOR gate, the expression may be further reduced:

$$P=(A\oplus B)\oplus(C\oplus D)$$

and the circuit formed is a 'cascade' of XOR gates:

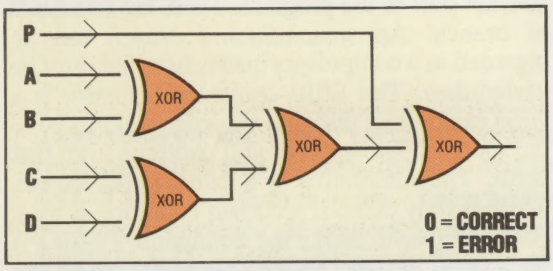


A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

PGB Truth Table

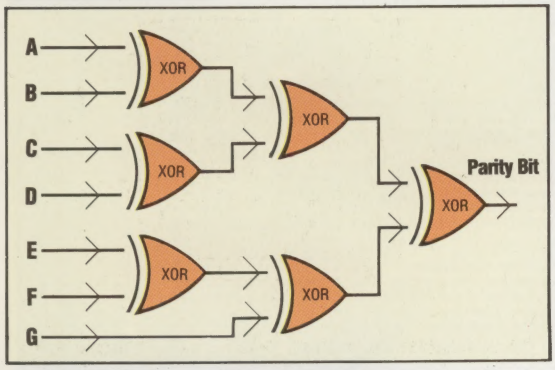
This circuit can be modified to act as a parity checker by simply adding another XOR gate to compare the received parity bit with one generated by the circuit at the receiving end.

Five-bit Received Code



In practice, most computers use the international standard ASCII code for data transmission. This is an eight-bit code with seven information bits and one even parity bit. It is easy to envisage a parity bit generator for ASCII codes:

Seven-bit Information Code



A PRIORITY ENCODER

Many computers use 'priority interrupt' systems to control the flow of data to and from peripheral devices. In these systems the CPU's operation is interrupted by a signal from the peripheral when it needs the CPU's attention. Where two or more peripherals interrupt the CPU at the same time, however, an order of priority must be followed for the CPU to 'service' the most important device first. The priority encoder that we shall design will link four peripheral devices into a circuit that can identify which peripheral is signalling, and will operate a priority system in the case of simultaneous signals from two or more devices.

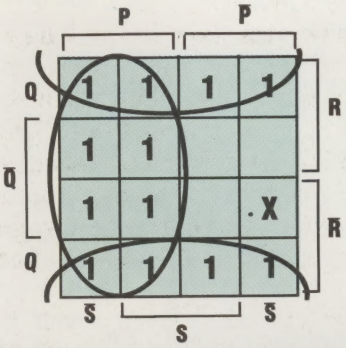
In order to output information specifying one of four devices, two output lines are required. In addition, a third output line will be used to signal that an interrupt is required. Let the four peripherals be P, Q, R and S; P having the highest priority and S having least. The output lines will be called A and B, to identify the peripheral, and Z to signal that an interrupt is required. The truth table for the encoder can be made using an X for conditions that the encoder 'doesn't care' about.

P	Q	R	S	A	B	Z
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

To help you to understand how this table is made up, look at the final row. In this case P is signalling an interrupt and as P is the device with highest priority we do not care whether or not the lower priority devices are signalling as well.

The three output lines from the circuit must be analysed independently. Starting with A, the k-map is:

For A



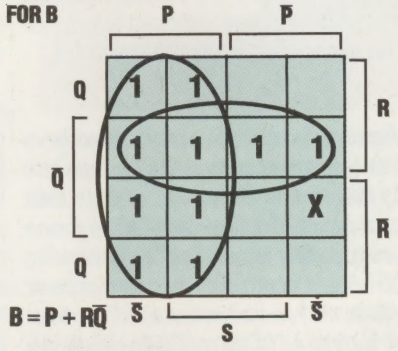
A = P+Q

The 'don't care' case on the output side for A is represented on the k-map as an X, but the 'don't care' cases on the input side are dealt with differently. Take the case where P is one and Q, R and S are 'don't cares'. Here we must fill in all the boxes on the k-map where P is one — there are eight in all. From the k-map we get the simplified expression:

$$A = P + Q$$

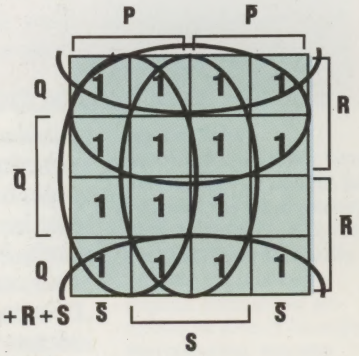
Similarly for B and Z, the k-maps are:

FOR B



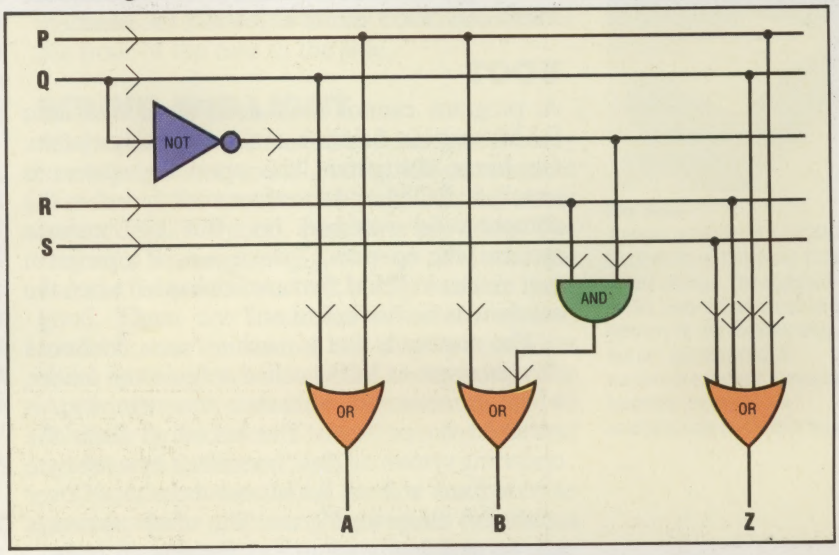
B = P + RQ̄

FOR Z



Z = P + Q + R + S

Using these three expressions, we arrive at this circuit design:



BIT

This word is supposed to have originated as a contraction of the term Binary digIT, though there is some suggestion that it may simply have derived from the American slang for a small sum of money, more familiar in phrases such as 'two bits'. It forms the root of a great many phrases in a contemporary jargon dictionary, and here are just a few of them:

Bit-copier: A utility program that can copy one disk to another, bit-for-bit. These programs have been used in pirating software, since they can usually overcome any in-built software protection methods.

Bit-error: A fault arising in a computer system when one or more bits in RAM memory or on magnetic disk flip from one logic state to another. These may be *soft* errors — caused by strong electrical interference or even cosmic radiation during periods of sunspot activity — in which case they should not recur when the computer is reset. Alternatively, they may be *hard* errors — for example, when a transistor in a RAM chip breaks down — in which case the whole unit will need replacing.

Bit-mapped: This refers to a microcomputer system where every dot or pixel on the display can be individually and independently switched on or off. Every pixel is represented by a bit in RAM memory.

Bit-twiddling (more correctly *bit-manipulation*): This is a term used among programmers for programming the computer at its lowest level.

BLOCK

A *block* is a fixed amount of data, usually a quantity of information stored on disk or cassette tape. It is usually the smallest amount of data that can be read from or written to disk at any one time. A system might, for example, have to write four complete 256-byte sectors at once whenever it writes to the disk. A block here would be four sectors — or one Kbyte — of data. Each block is a self-contained unit and often has extra error-checking information included in it so that the micro can detect errors as it loads the block.

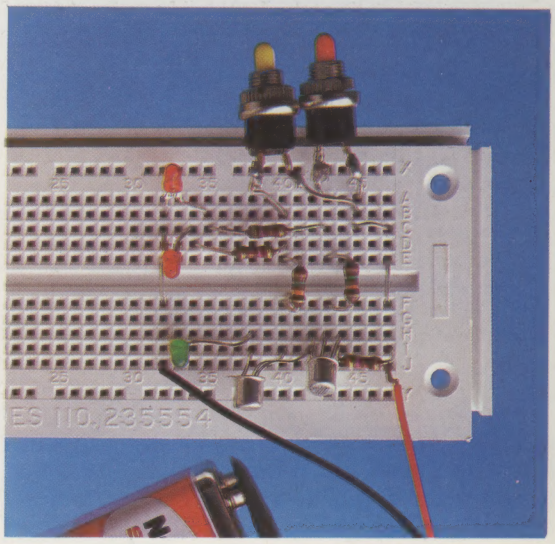
BOOT

A program cannot load itself from disk into RAM: that is a function of the operating system. On home computers, the operating system is usually in ROM, and therefore available when the computer is switched on. But on business systems, the operating system is itself a program that resides in RAM. So how is it loaded when the machine is first switched on?

The answer is that a machine must contain a tiny program in ROM called a *bootstrap loader*, which is activated whenever the computer is switched on or reset. The bootstrap finds the operating system on disk, transfers it byte-by-byte into a fixed area of RAM and then hands over control to the new program. The whole process is known as *booting up*, or just *booting*.

BRANCH

A *branch* instruction is one that directs the computer to stop executing the instructions in a program in the normal sequence and go to another part of the program. There are two types of branch. An *unconditional branch* can be regarded as a compulsory instruction that must be undertaken. The GOTO command in BASIC is a good example: it instructs the interpreter to go to the specified line number and continue from that point. The alternative is a *conditional branch*, which is only taken if a certain condition is true. For example, in BASIC, the command IF SUM>4 THEN GOTO 100 will branch to line 100 only if the value of SUM is larger than four. Quite often, people distinguish between these two by using the word *jump* for an unconditional branch and *branch* for a conditional branch.



BREADBOARD

This is the name given to a device onto which electronic components can be mounted for the purposes of experimentation and circuit development. *Breadboards* come in two main forms. The first of these is really a kind of general-purpose printed circuit board — a stiff piece of card drilled with an array of holes and printed on one side with parallel copper tracks. Integrated circuits and discrete components can be soldered onto the board, then the tracks may be severed with a sharp knife, or joined using a piece of wire as a bridge, to form the correct interconnecting circuit between components. The result is often quite untidy, but it is a good deal quicker and cheaper than designing and etching a printed circuit board for each experimental phase of a product.

The main alternative to this type is the solderless breadboard, which is intended for home experimentation rather than professional use. It is typically made from plastic and is around 1cm thick. The connecting wires run inside the casing, and electronic components can be pushed into the small holes in the top surface and pulled out again when the circuit is to be changed.

ATARI UPDATE

Atari entered the home computer market very early with two models, the 400 and the 800. These computers were characterised by a high standard of construction, superb graphics and excellent sound. Now Atari has updated both models. The two new XL models sell at a cheaper price and include some welcome refinements.

Faced with the growing competition among home computer manufacturers, Atari redesigned its line of computers and introduced the 600XL and 800XL. The new machines can use the software written for the 400 and 800, which opens up a wide range of programs, from top-selling games to business packages.

Fortunately, both machines have adopted the keyboard of the 800. The Atari 400 has a membrane-style keyboard, similar to the ZX81, with its characters printed on a flat plastic membrane. Because of the small size of the keys and the amount of pressure needed to make contact, touch-typing is virtually impossible. In contrast, the 800 has a full-size typewriter-style keyboard that is one of the best on the market.

The keyboards of the new machines are identical and have a total of 62 keys. These include four function keys: START, SELECT, OPTION and RESET, which are positioned on the right-hand side of the keyboard. There is also a HELP key, which works with some new software to provide helpful hints on the screen. There are 29 graphics keys and the Ataris have a full ASCII character set built in. One quirky feature of the keyboard that has been inherited from the 800 is the way the cursor keys are used. The arrows are printed as the third character on four separate keys. To move the cursor you hold down the CONTROL key while pressing the appropriate arrow key. Apart from this strange feature, the keyboards are well-designed and comfortable to type on.

The Atari 400 comes with 16 Kbytes of user memory, while the 800 has 48 Kbytes. Both computers have expansion slots on the system board that can be accessed by removing the top of the machine. The XL models should not be opened. Instead, expansion ports are built into the outside of the case. The only significant difference between them is the amount of memory that comes as standard. The 600XL comes with 16 Kbytes, which can be expanded to 64 Kbytes by plugging in an expansion pack. The 800XL comes with 64 Kbytes.

Both of the XL machines have an interface built in called the 'Expander'. This is a bus-type expansion port that can be connected to a variety of peripherals and expansion packs. There is also a ROM cartridge slot just behind the keyboard for cartridge games and other software. The original Atari 800 has two cartridge slots, but virtually no software has been written to make use of the second slot, so it has been removed from the XL line. The 400 and 800 have four joystick ports, but again there is little software that uses all of these, so they have been reduced to two on the XL machines, and these ports have been moved from the front of the case to the side.

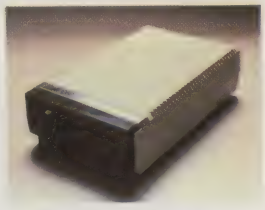
IAN McKINNELL

The Atari Twins
Atari's 600XL and 800XL home computers are very similar, but the 600XL has 16K of memory while the 800XL has 64K. The machines have a high quality keyboard and good colour graphics. Because they are updated versions of the original Atari computers, they also enjoy a wide range of software

SOUND AND LIGHT

All the Atari computers can be connected directly to a television set, and all but the 400 can connect to a computer monitor as well, but the Atari screen display is good, even on a television. The character set is generally easy to read and the contrast between text and background is quite good. There are a number of colour options available, but the normal text display comprises white lettering on a deep blue background. The Atari computers display a maximum of 40 columns by 24 lines of text. There are four text modes with different displays.

The Ataris were among the first computers to provide sprite graphics. Their sprite function is called Player-Missile graphics, and is controlled



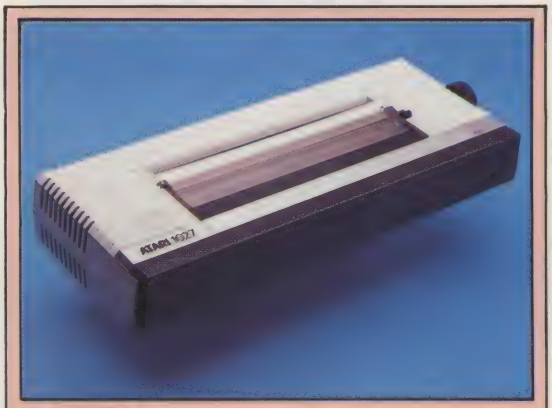
Disk Drive
A useful option for the 800XL is the disk drive. This gives 127K of fast storage. The standard 600XL does not have sufficient memory to use the disk drive but can be expanded. A number of excellent games and business programs are available only on the disk format

by a special chip in the computer called the GTIA chip. 'Players' are objects created from pixels. Once the shape of a player has been determined, the values of each pixel are POKEd into an area of memory called a 'shape table'. You can create as many as four players, each with an associated missile component. The player is assigned one or more colours, and is manipulated on the screen by changing the values in the shape table. Though Atari's Player-Missile graphics are not easy to use, they provide some remarkable screen displays.

The new Atari machines have 11 graphics modes, and up to 256 colours (actually 16 colours, each with 16 different shades); because of the amount of memory required for screen display, however, the number of colours that can be shown varies according to the resolution of the screen. The higher the resolution, the fewer the colours that can be displayed. The maximum graphics resolution on the 600XL and 800XL is 320 by 192 pixels.

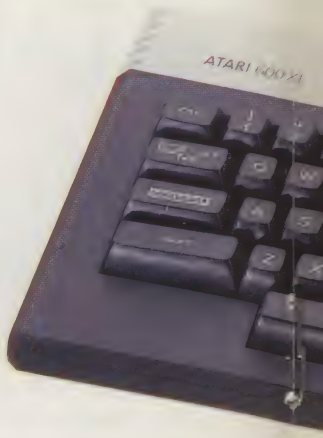
Atari's sound features are also controlled by a specially designed chip. There are four independent voices, each of which has a range of $3\frac{1}{2}$ octaves. The voices can be controlled through the SOUND command in BASIC, or by POKEing values into the memory registers that produce the various tones. Tones can be adjusted for oscillation, pitch, distortion and volume. When the voices are controlled with SOUND, only one voice can be started at a time. This means that harmonising requires you to turn on each voice separately, and there is a noticeable delay. This problem can be overcome by using machine code routines, or using POKE in place of SOUND.

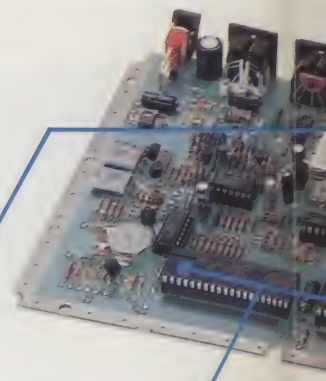
The Atari 400 and 800 have no language built in; you have to use a separate cartridge. The 600XL and 800XL have Atari BASIC built in. This is not the best BASIC, lacking in many of the features that make BBC BASIC and others so good. For instance, there is no CIRCLE command, no PRINT@ or PRINT USING feature, no automatic line numbering or renumbering, and no provision for integer variables. However, Microsoft BASIC and Extended BASIC are available in ROM cartridge.



Atari 1027 Printer
This printer has a print head similar to the golfball used in some typewriters. It gives a print-out as good as most electric typewriters, but is rather slow in operation.

There are two other Atari printers, one using ballpoint pens to draw letters and lines in four colours, and a dot matrix printer that gives a poorer quality print-out but is fast. These are the only printers that can be used directly with the XL machines because of the lack of a standard printer interface





Cartridge Port
The XL range has a single slot for ROM cartridges

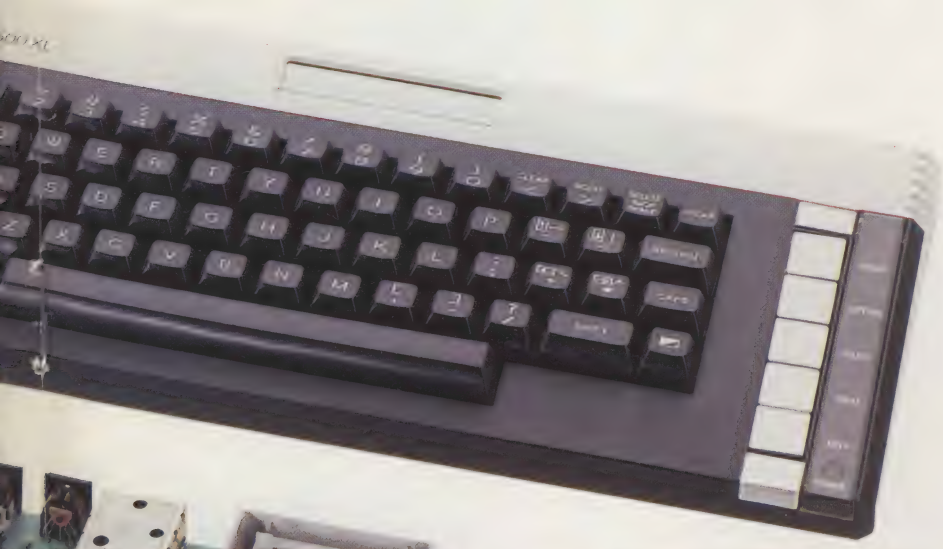
Graphics Chips
Two custom-built chips, known as ANTIC and GTIA, provide the Atari's spectacular display abilities

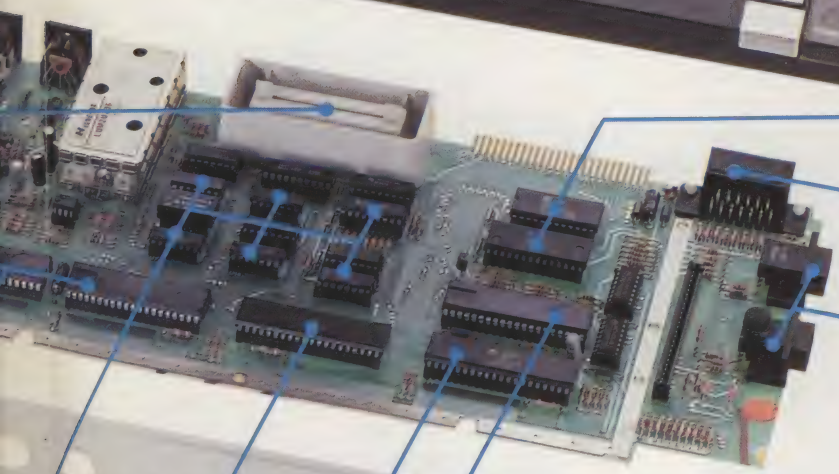
RAM
A number of chips comprise the 16K of RAM

Ball And Stick
The Atari offers two main ways of controlling games. The conventional method is with a joystick and the Atari joystick (shown right) has become something of an industry standard. The Atari trackball (left) gives control through a ball that spins in its mount

To accompany the new line of computers, Atari has redesigned the existing peripherals and added to the number of options available for expansion. Perhaps the most useful peripheral is an expansion box, which plugs into the Expander. The expansion box provides eight expansion slots, which can hold interface cards for several peripherals, two RS232 serial ports and a parallel bus. Atari also has a CP/M module with a Z80 microprocessor, the CP/M 2.2 operating system, and switchable 40/80 column display.

Atari have scored high marks with the 600XL and 800XL in design, quality of construction, and features. They also have the advantage of a huge library of software on cartridge, cassette, and disk that has grown up over the last few years. These two new machines from Atari should prove a very attractive buy for home computer enthusiasts.





ROM
Two ROM chips hold the BASIC interpreter

Peripheral Port
A 13-pin port is used to connect the peripherals, including disk drives, printers, and the dedicated cassette recorder

Joystick Ports

Sound Chip
A custom-built chip called POKEY handles sound generation

I/O Chip
A 6520 handles the input and output ports

CPU
The Atari is based on a fast 6502 chip

Atari Cassette Recorder
The Atari only works with its own cassette recorder. Although this raises the price of the system, it does have advantages. Firstly, because the recorder is made by Atari it is more reliable. Secondly, the recorder uses two tracks. One track is for saving programs in the normal way, the other can have sound recorded on it. This allows language teaching programs to play fragments of speech at just the right moment

ATARI 600XL/ 800XL

PRICE
600XL:£160
800XL:£250

DIMENSIONS
600XL:380×170×40mm
800XL:380×220×40mm

CPU
6502, 2MHz

MEMORY
16-64 Kbytes RAM, 24K ROM

SCREEN
Up to 24 rows of 40 columns of text, graphics up to 320×192 with sprites and 16 colours in 16 brightness levels

INTERFACES
Joysticks (2), peripheral port, expander port, cartridge port

LANGUAGES AVAILABLE
BASIC, FORTH, LOGO, PILOT, 6502 Assembly language

KEYBOARD
Typewriter-style with 62 keys, including cursor keys and dedicated function keys such as SELECT and START for program control

DOCUMENTATION
Manuals have never been one of Atari's strong points as the company has tended to view its computers primarily as games machines and has skimped on technical details. However, there is a vast range of superb independent manuals and magazines, although these do add to the cost of owning an Atari

STRENGTHS
The Ataris continue to offer the best in games computers with superb sound and graphics, and a vast range of software to choose from. The new machines also provide excellent expansion options

WEAKNESSES
Ataris can be expensive – you need a dedicated cassette recorder and the price of software is unusually high. Graphics and sound programming are also more difficult than with many machines

LINES OF ENQUIRY

In the last instalment of our business series we looked at how certain packages handle stock control. An efficient program needs to monitor all stock movement, from supplier to the shop shelves. The packages achieve this by a system of coding structures. We now look more closely at the design requirements of these structures.

We have discussed in some detail the different ways stock lines can be identified in the stock data file. In addition to simply allocating them a code number, however, the system should allow the user to record information on each stock line.

This information will need to be categorised into relatively constant data and the amount of data will be determined by the storage capacity of your computer. The design of such a system will therefore have to be very economical. The amount of information must be sufficient to meet basic management needs, but too much information will make impossible demands on the computer's processing and storage power.

Stock Answers
The Stock Recording System is one of the many Dragon 64 applications programs written to run under OS9. This is a multi-programming and multi-tasking operating system developed by Dragon from the UNIX operating system

IAN McKINNELL

Dragon Data's Stock Recording System for the Dragon 64 has seven *fields* in which to record information for each stock record. These consist of an item number, description, re-order level, cost price, sale price, and the unit of measure.

The fields form an important structure in a stock system. The difference between the cost price and sale price will provide a measure of gross profit. Items can also be grouped, which helps both to analyse and summarise in reports. The unit of measure field is essential because commodities are packaged in varying ways. On one stock line the owner will want to count each individual item, on others, such as nails and screws, the count will be by the box.

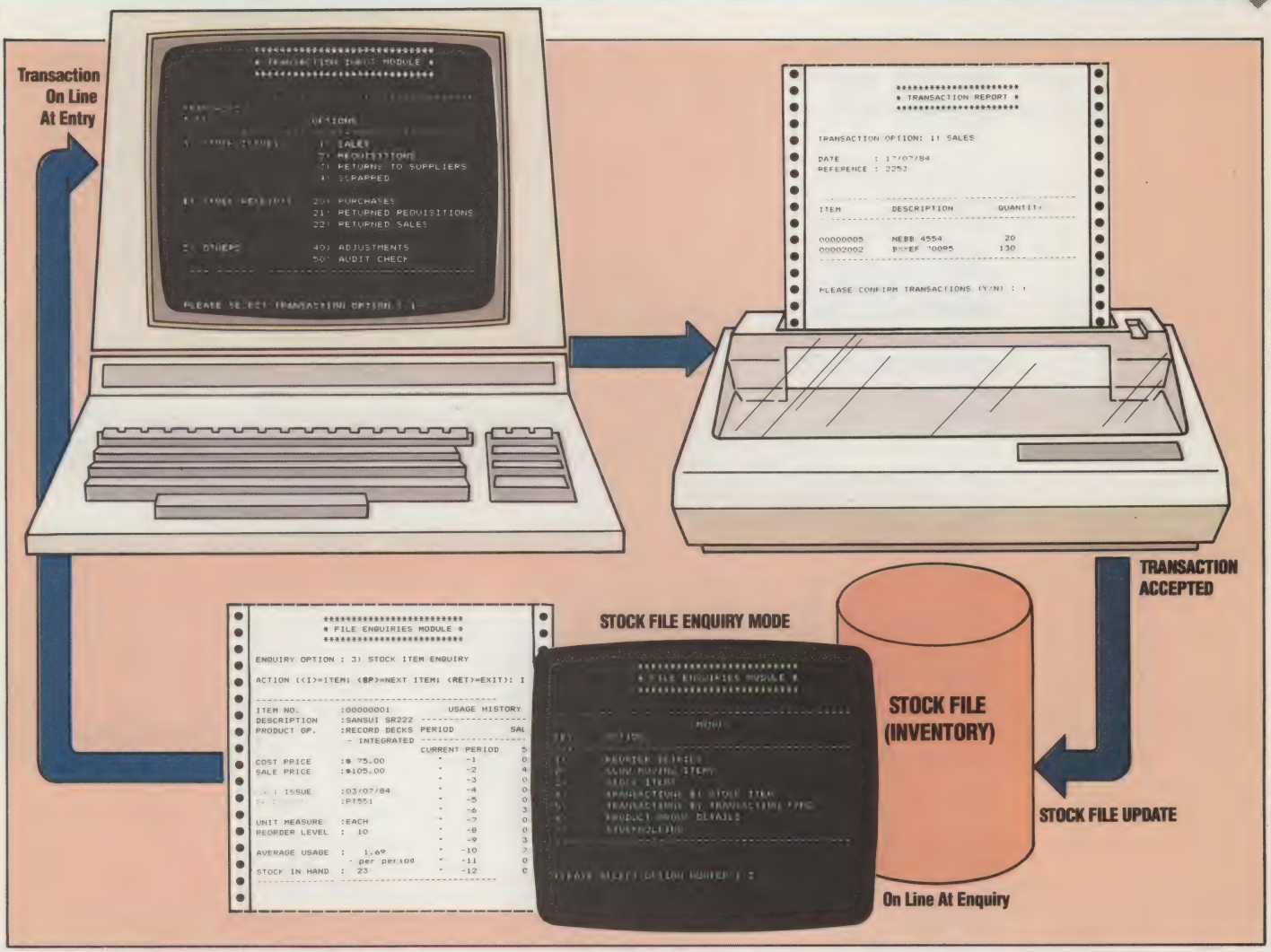
Perhaps the most fascinating aspect of designing a stock control system is that much of the information required consists of *dynamic* rather than *static* data. We have already seen that standard ledger records consist of relatively constant information about the customer or account (known as the header part of the record) and information about the various transactions taking place on that account.

With stock control, however, the firm boundaries that exist between the static part of the record and the dynamic or transaction-based part of the record become much less clearly defined. The stock item and group descriptions and coding are the equivalent of the static information on the customer or supplier records in sales or purchase ledger systems. But with a stock control package, static data has to be supplemented by a great deal of information derived from stock transactions.

For example, customers on a sales ledger master file change their address or telephone number relatively infrequently. Therefore, the program will have a master file maintenance option that will allow you to amend particular client records.The information or amendment cannot be calculated by the program.

If we take the sales price and cost price fields on the stock item record as an example, these can change each time the business replenishes its stock. The logical solution here is for the program to take this information (together with the date of the price change and the stock volumes affected) directly from a transaction input routine recording goods received, and not from a file maintenance program. There will still be, of course, a need for a file maintenance program to amend stock descriptions, or add or delete a stock line from the stock file, but a lot of data can be generated from basic transaction information.

In order to understand how a stock control program operates, we need to look at the input



routines and the enquiry facilities and reports that these lead to. We have taken the Dragon's program as an example and have illustrated the various files in our diagram. The three important elements are: the transaction input routines, the transaction details and the stock item enquiry file.

The transaction input displays all have the same layout in the Dragon system. The transaction types are more or less self explanatory. We will concentrate for the moment on sales (i.e. on movements out of the stock register). But first it is worth noting that all business programs are designed to be as 'friendly' as possible, and will prompt the user to enter all the necessary information. This leads to two different programming requirements that have to be met if the system is to be successful.

On the one hand the program has to recognise certain data fields and perform arithmetical or other operations on the data. On the other hand the program has to guide the user and recognise inappropriately entered data. In other words, it has to perform checks on the data entered.

Once the user has entered the data, the reference number to identify the authority for the entries (the same number would be marked on the sales invoices from which these entries would be generated), and the item number, the computer will read the stock file to see if such an item exists. If it finds the number, it will automatically display the description allocated to that item number. This acts as a visual check for the user, who can then enter the quantity sold.

From this information, the computer is able to extract a large number of enquiries and reports. For example, one of the options on the main menu, FILE ENQUIRIES, has a sub-menu consisting of seven options covering stock items, transactions and product groups. These are: STOCK DETAILS, SLOW MOVING ITEMS, RE-ORDER DETAILS, TRANSACTIONS (BY STOCK ITEMS), TRANSACTIONS (BY TRANSACTION TYPE), PRODUCT GROUP DETAILS and STOCKHOLDING.

The sales transactions affect all these reports. If a re-order report is requested, for example, the program will check to see if the stock sold on these two items has taken the quantity of stock in hand below the specified re-order level.

Every detail entered on the sales transaction display is relevant and is used in some way. The Dragon program enquiry display provides a clear illustration of how much management information can be derived from those sales transactions. The usage history gives an immediate indication of the speed and volume of commodities moved over the year, the average usage period consolidates this, and the last issue date and reference are also displayed.

Authorised Entry
The stock control program accepts transaction information, and checks the user's authority to enter information and that the transactions are valid. The stock file, which contains the individual stock item records, is on line during this process, and is updated by it. The program also contains a database module, allowing the user to inspect the stock control file and create reports on various aspects of the inventory

TESTING TIME

In the previous instalments of Workshop, we investigated many of the ideas and techniques used in the construction of computers and their peripherals. Before proceeding to undertake more complex projects for your home computer, we now pause for a brief overview of the work covered so far.

We started the course by suggesting that the right tools for the job are essential (see page 44). A range of equipment is desirable for the specific tasks involved in construction, alteration and repair work. You *can* use a flat head screwdriver on a cross head screw, but that is likely to lead to damage to both of them. You *can* use a pair of pliers to undo a nut, but a spanner will do the job quicker and without ruining the nut. In the long run, it pays to invest in a set of tools specifically designed for these jobs.

Attaching and joining wires is the same. A good solder joint will last longer and operate better than one that has been hastily connected. Always apply the solder to the wire, not to the iron, as the solder will slowly corrode away the tip of your iron. The less contact the two have, the better. Let the solder flow over the two wires you want to join, and only when the whole joint is covered, remove the heat and cool the joint by blowing on it gently. Keeping to these rules will ensure the joint is not dry. Desoldering allows you to remove components from circuit boards safely for repair or replacement (see page 68).

We have discussed the fundamental concepts of digital electronics, both in Workshop and in the Computer Science course. We have introduced the basic components and seen how these interact. The resistor is the simplest of all the components, but also the most frequently used. If you look inside your microcomputer, you will see more resistors than integrated circuits.

The capacitor is also very common. In computers, these are used to filter off the unwanted noise that attaches itself to a signal. Every time that a signal is changed, amplified, or otherwise used, it is degraded. So a means of restoring the signal to its uncorrupted form is essential. The capacitor is the simplest way of removing the unwanted elements of a signal.

The most important component that we've discussed, however, is the transistor. This is the essential component of all the devices that are used to change and manipulate the signals in a computer. The transistor can be used to amplify a signal, or to switch signals on and off. Most importantly for the computer, the transistor can switch a signal on and off in accordance with one or several other signals. This is what goes on inside a logic gate.

We have built the three simplest logic gates, NOT, OR and AND, from a handful of components (although individual gates within the computer's integrated circuits are often made up of other, more complex types of transistors).

Logic gates are not particularly useful in themselves, but they can be combined into logic circuits that can perform operations on data. In the last instalment, we constructed a half adder — a simple logic circuit to add two binary bits — from the logic gates on two integrated circuits.

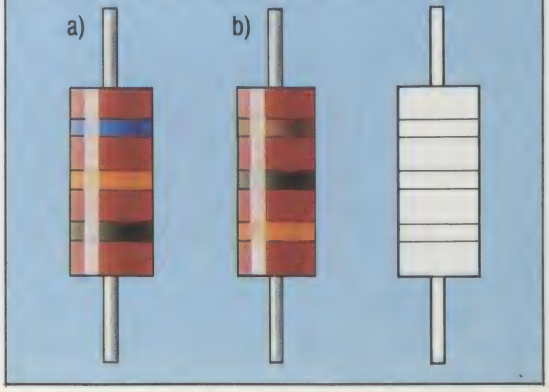
Integrated circuits are some of the most complex components used in electronics. The simple transistor-transistor logic (TTL) chips that you have used so far are small scale integration (SSI) packages. There are only a few transistors in each chip. These were the first type of chip made, and early computers relied upon them. As techniques improved, however, so the number of transistors that could be fitted onto a single chip increased. Medium scale integration (MSI) allowed complete logic circuits to be available on a single chip. An example of these is a full adder — equivalent to two of the half adders we built (see page 165) combined.

Large scale integration (LSI) and very large scale integration (VSLI) have also been developed. A complete CPU on a single chip is an example of VLSI. As there are literally thousands of transistors inside a microprocessor, it is difficult to imagine the logic circuit of such a chip, but this complexity makes individual chips very powerful and easy to design into circuits.

Having mastered these fundamentals, we can proceed to tackle more complex ideas, and this should enable you to construct some useful additions to your micro. First, though, try your hand at the projects opposite. None of them is very complex, and none requires more than a couple of transistors or integrated circuits.

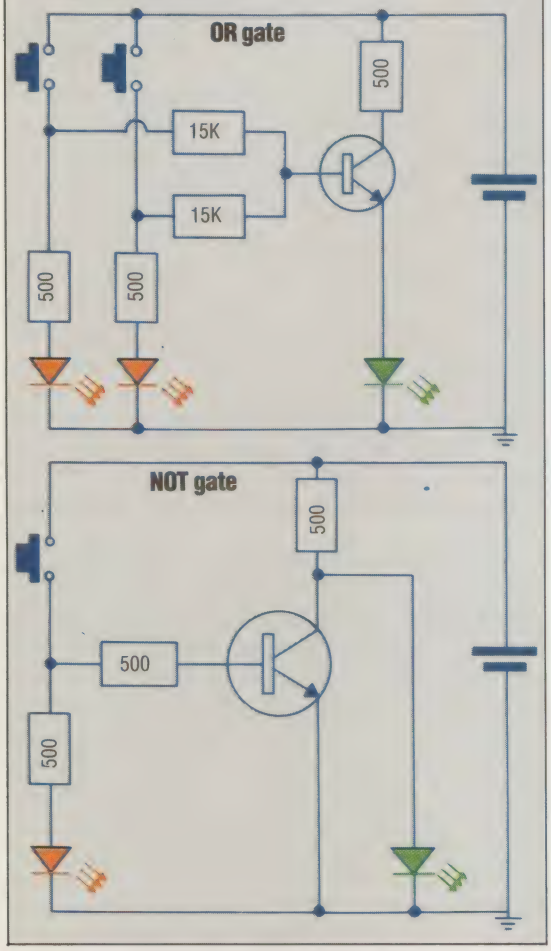
1) Resistors

Resistors have a series of coloured bands around them to identify their value. What are the resistances of the two resistors shown here? What would the colour bands be for a 150-ohm resistor?



2) NOR Gate

On page 145, we constructed NOT, OR and AND gates using transistors. This first practical exercise is to build a NOR gate using a similar circuit to those for NOT, OR and AND. As a helpful hint, the circuits for OR and NOT appear below. There are two approaches to this problem. You could use your knowledge of logic circuits to construct a NOR gate by combining an OR and a NOT gate. Alternatively, you may spot a short-cut method if you study the NOT gate circuit closely.



3) Decimal To Binary Converter

Construct a circuit that converts decimal to binary. In order to keep the circuit simple, we'll restrict this exercise to two-bit binary numbers, that is, decimal numbers from zero to three. Your circuit should have four input switches labelled zero, one, two and three. When you press one of these switches, the corresponding two binary bits should be shown on a pair of LEDs. A partial truth table for the converter looks like this:-

ZERO BUTTON	ONE BUTTON	TWO BUTTON	THREE BUTTON	HI-BIT	LO-BIT
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

4) BCD Or Not BCD?

We have studied the binary-coded decimal (BCD) numbering system in the Computer Science course. This is a numbering system halfway between decimal and binary — each decimal digit is converted to its binary equivalent and the resulting four-bit groups are joined together to make the BCD number. For example, 53 is coded as 01010011, 5 is represented by 0101 and three is represented by 0011. This means that any valid BCD digit is a group of bits from 0000 to 1001 corresponding to zero to nine in decimal. Any codes in the range 1010 to 1111 are illegal in BCD.

Build a circuit to test whether a given four-bit input is a legal digit. Your circuit should have four switches: B0, B1, B2 and B3, representing the number being tested. There should be two outputs: a green LED if the number is a legal BCD digit and a red LED otherwise. The truth table looks like this:

Decimal Equivalent	B3	B2	B1	B0	Valid BCD	Invalid BCD
0	0	0	0	0	1	0
1	0	0	0	1	1	0
2	0	0	1	0	1	0
3	0	0	1	1	1	0
4	0	1	0	0	1	0
5	0	1	0	1	1	0
6	0	1	1	0	1	0
7	0	1	1	1	1	0
8	1	0	0	0	1	0
9	1	0	0	1	1	0
10	1	0	1	0	0	1
11	1	0	1	1	0	1
12	1	1	0	0	0	1
13	1	1	0	1	0	1
14	1	1	1	0	0	1
15	1	1	1	1	0	1

From this truth table, we can see that the BCD signal is given by $\overline{B3}+\overline{B2}.\overline{B1}$. The invalid BCD signal is the NOT of this:

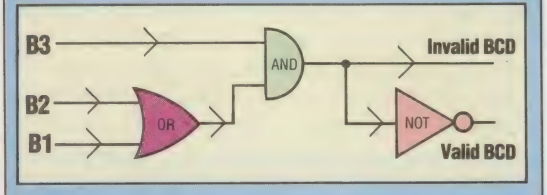
$$\overline{\overline{B3}+\overline{B2}.\overline{B1}}$$

which can be simplified as:

$$\overline{\overline{B3}} \cdot \overline{\overline{B2}.\overline{B1}}$$

$$B3.(B2+B1)$$

So the circuit to test for an invalid BCD digit is quite simple and only involves three of the inputs. A suggested logic diagram for the BCD validator circuit is:



Answers in the next part of Workshop.

MODES OF ADDRESS

The strength and versatility of Assembly language instructions is enhanced by the variety of ways in which these instructions can address memory. The different ways consist of direct, indirect and indexed addressing, and combinations of these. In this instalment we take a closer look at modes of addressing in machine code.

Every Assembly language instruction refers explicitly or implicitly to the contents of memory, and since one byte is distinguishable from another only by its address, every Assembly language instruction must, therefore, refer to at least one address. The manner in which an address is referred to may be direct and obvious, as in LDA $E349, which means 'load the accumulator with the contents of the address $E349'. In this case, both the accumulator (a byte with a name rather than a number for its address) and the address $E349 are mentioned unambiguously, and the nature of the relationship between them is clear.

On the other hand, the reference to an address may be much less obvious: RET, meaning 'return from a subroutine call', is a good example. This may not seem to refer to an address at all until you expand it into 'the location address of the next instruction to be executed is the place where the last subroutine call was made'. Here, the address whose contents are to be changed (i.e. the program counter — the register holding the address of the next instruction to be executed) is not mentioned, nor is the address at which its new contents (i.e. the new location address) are to be found. These two instructions can be seen as highly contrasting examples of *addressing modes.*

In the course so far we have seen instructions in two kinds of addressing mode: *immediate mode,* as in LD A,$45 or ADC #$31, and *absolute direct mode,* as in STA $58A7 or LD ($696C),A. These may seem like the 'natural' addressing modes, covering every possible case except the implicit modes such as RTS or RET, but there are other possibilities as well. We shall look at these separately.

ZERO PAGE ADDRESSING

Zero page addressing (also known as *short addressing*) is used whenever an instruction refers to an address in the range $0000 to $00FF. All addresses in this range have a hi-byte of $00, and therefore lie on page zero of memory. Such instructions need only two bytes — one byte for the op-code and one for the lo-byte of the address. When the CPU detects a single-byte address at a point where it expects there to be two bytes, it assumes a hi-byte of $00, and so refers automatically to page zero. The advantage of this page zero mode is speed of execution: data on page zero is accessed significantly faster than data on any other page precisely because it requires only a single-byte address.

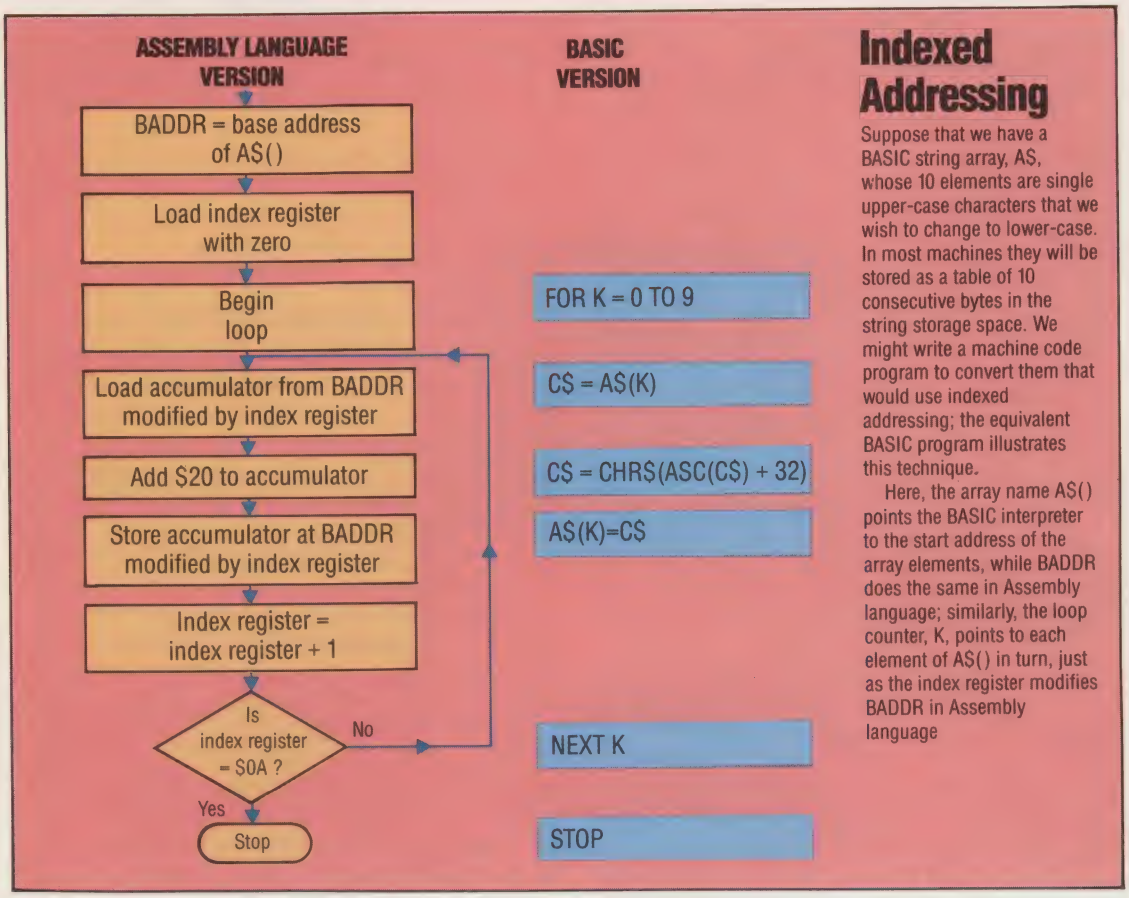
The Z80 and the 6502 microprocessors differ greatly in their use of the zero page mode. In 6502 Assembly language, any instruction that addresses RAM (such as LDA) can be used in the zero page mode, and all the 256 bytes of zero page are available to the CPU. In Z80 Assembly language, only one instruction — RST (the 'restart' or 'reset' instruction) — features the zero page mode, and it can address only eight specific page zero locations. Because the RST instruction is so specific in its addressing, it requires only a single-byte op-code (the address forming part of the op-code itself), which makes it very fast in execution. We will see more of the Z80 RST instruction later in the course because of the special uses to which it is put.

It may seem ridiculous to be comparing the speed of Assembly language instructions when the execution time of the slowest instruction is measured in microseconds anyway, but it isn't difficult to write Assembly language programs in which saving a microsecond per instruction execution can mean the difference between success and failure. Games programs featuring animated high-resolution three-dimensional colour graphics, for example, can involve millions of instructions per screen operation, and they must execute these commands as quickly as possible for the sake of smooth animation. Shaving a microsecond off one operation becomes very important when you put that operation inside a loop and execute it 64,000 times consecutively.

INDEXED ADDRESSING

Indexed mode is vital to Assembly language programming since it permits the construction of array-type data structures. Without such structures, programs are restricted to addressing memory locations individually by address: this is what we have done with all the programs so far in the course. Once indexing is possible, however, far more data can be handled, and more power is put at the programmer's disposal.

The essential elements of indexed mode are a *base address* and an *index.* Suppose we wish to keep a table of data — ASCII character codes, for example — in consecutive bytes. The base address of the table is the address of its first byte.


ASSEMBLY LANGUAGE VERSION
BADDR = base address of A$()
Load index register with zero
Begin loop
Load accumulator from BADDR modified by index register
Add $20 to accumulator
Store accumulator at BADDR modified by index register
Index register = index register + 1
Is index register = $0A ?
No
Yes
Stop
BASIC VERSION
FOR K = 0 TO 9
C$ = A$(K)
C$ = CHR$(ASC(C$) + 32)
A$(K)=C$
NEXT K
STOP
Indexed Addressing
Suppose that we have a BASIC string array, A$, whose 10 elements are single upper-case characters that we wish to change to lower-case. In most machines they will be stored as a table of 10 consecutive bytes in the string storage space. We might write a machine code program to convert them that would use indexed addressing; the equivalent BASIC program illustrates this technique.
Here, the array name A$() points the BASIC interpreter to the start address of the array elements, while BADDR does the same in Assembly language; similarly, the loop counter, K, points to each element of A$() in turn, just as the index register modifies BADDR in Assembly language
LIZ DIXON

Given this, we can refer to every subsequent byte in the table by its position relative to the base address, so that the first byte is in position zero, the second byte is in position one, the third in position two, and so on. A byte's position relative to the table base address is called its index, and the absolute address of any byte in the table is calculated from the sum of the base address and the byte index. If we can construct a program loop in Assembly language, and use the loop counter as an index to the base address of the table, then we can address each byte of the table in sequence, just as we might access the elements of a BASIC array using a FOR..NEXT loop.

Once again, the Z80 and 6502 Assembly languages handle indexed addressing differently. The 6502 chip contains two single-byte registers called X and Y, each of which can hold an index that modifies a base address. This limits the length of a table to 256 bytes (the largest possible single-byte number). The Z80 chip contains two two-byte registers, IX and IY, which may hold the base address itself, and can then be incremented or decremented to point to successive bytes of the table. Since they are two-byte registers, IX and IY can address any byte addressable by the CPU itself. Their contents can also be modified by a single-byte index.

INDIRECT ADDRESSING

Indirect addressing involves the use of pointer addresses, a concept which was introduced early in the course, in relation to floating boundaries in memory (see page 58). Imagine that a group of people form a cinema club and that they meet every week to watch a film chosen by the club president. The film may be showing at any one of a dozen different cinemas, so when he has chosen the film for the week, the president writes details of the time and place on a postcard which he then sticks in the window of a shop in the centre of town. Club members don't know where the film will be from week to week, but they know where the shop is, and the shop 'points' them to the correct cinema. The address of the shop is, indirectly, the address of the cinema.

In indirect addressing mode it is possible to

Indicator Pointer
Examples of indirect addressing do not often appear in everyday life. However, in this photograph the train indicator board contains the actual data wanted by the traveller, so the sign telling him where to find the board indirectly addresses that data. Indirect addressing in an Assembly language instruction means that the address supplied in the operand contains the address of the byte where the data is stored; the operand address is a pointer

write instructions that contain the address of a pointer, and which operate on the contents of the location that the pointer indicates (not on the contents of the pointer itself). The advantages of this addressing mode are considerable, especially when combined with the indexed mode. Suppose, for example, that you write an Assembly language routine that searches a data table for a given character, and returns with the index position of the character. Suppose, also, that you want to keep several such tables in different places in memory, and that you want to use the same routine for searching any of them. If the routine is written so that it finds the base address of the search table indirectly via a given pointer location, then it can be used on any table provided that the contents of the pointer location are properly adjusted before the routine is called.

In general, programs require mixtures of these modes rather than pure examples of single modes. The 6502 instruction LDA, for example, can be used in the following modes:

Op-code	Operand	Mode
LDA	#$34	Immediate Mode
LDA	$A2	Zero Page Direct
LDA	$967F	Absolute Direct
LDA	$A2,X	Zero Page, Indexed by X
LDA	$967F,X	Absolute, Indexed with X
LDA	$967F,Y	Absolute, Indexed with Y
LDA	($A2,X)	Indirect, pre-indexed with X
LDA	($A2),Y	Indirect, post-indexed with Y

It is convenient to use a 6502 instruction in this example because it shows the combinations of addressing modes so clearly. Notice that the two indirect versions of the instruction are in zero page as well as indirect and indexed modes. A table such as this may seem confusing at first sight, but actually using the various modes soon makes their significance clear, and so far we have used both LDA and ADC in two modes — immediate and absolute — without confusion.

The table does answer the question posed in the last instalment of the course — how to tell the addressing mode of an instruction when the mnemonic is the same in every case? It can be seen that the format of the operand is different for every mode, and that the only ambiguity possible is whether an instruction such as LDA SYMB1 is zero page or absolute mode. An assembler program will resolve this for you, but if you are assembling the program by hand, you will have to determine whether SYMB1 has been defined as a single byte or as a two-byte quantity.

In general, once you start using an assembler, you can forget about things like op-codes and the number of bytes per instruction, and concentrate on learning the programming techniques of Assembly language. It is important to understand the mechanics of machine code, but Assembly language used with a good symbolic assembler is a much better way of programming, combining the power of machine code with many of the facilities of high-level languages.

Answers to Exercise On Page 178

The Monitor program on page 118 was written in modules with expansion in mind, so adding a menu option is reasonably easy:

SPECTRUM VERSION

1) Adjust the initialisation by editing or adding the following lines:

```
1050 LET LT=5:DIM C$(LT):DIM 0$(LT,24):DIM
     X$(16)
1150 LET C$="ADGQB":LET C1=-48:LET
     C2=10-CODE(C$(1))
1280 LET 0$(5)="    B - BINARY DISPLAY"
```

2) The input routine at line 2000 has already elicited the start address, standardised it as a four-digit hex number in A$, and converted it to a decimal number in DN, so the binary display subroutine reads:

```
7000 REM**HEX&BIN DISP S/R**
7020 FOR P=DN TO (DN+15)
7040 LET NM=P:GOSUB 3100: PRINT H$,
7060 LET N=PEEK(P): LET NM=N
7080 GOSUB 3000: PRINT B$;" ";
7100 GOSUB 7300: PRINT B$
7120 IF P=65535 THEN LET P=DN+15
7140 NEXT P
7200 RETURN
7300 REM**BINARY BYTE S/R**
7310 LET B$=""
7320 FOR D=8 TO 1 STEP-1
7330 LET N1=INT(N/2)
7340 LET R=N-2*N1
7350 LET B$=STR$(R)+B$
7360 LET N=N1
7370 NEXT D
7380 RETURN
```

BBC/COMMODORE VERSION

Copy the Spectrum version, with the following amendments:

1) Change the initialisation of LT and 0$() as in the Spectrum version above, and add C$(5)="B" to line 1150.

2) Line 600 transfers control to the command routine, so on the Commodore 64 and BBC Micro change this to:

```
600 ON CM GOSUB 5000,5500,6000,6500,7000
```

3) On the BBC change line 7060 above to

```
7060 N=?(P):NM=N
```

4) On the Commodore 64 change line 7350 above to:

```
7350 B$=MID$(STR$(R),2)+B$
```

BRANCHING OUT

In the past few years, Tandy's name has become a familiar sight in the high street. But the company's development spans over 50 years, from being a small supplier of electrical components to one of the world's largest retailers of microcomputers, as well as stocking a wide range of other domestic electronic goods.

The Tandy Corporation, through its retail divisions, Tandy and the American Radio Shack chain, has 392 computer centres and over 5,500 retail outlets in 76 countries. The company owns 29 factories that supply equipment for sale under the Tandy and Radio Shack brand names.

Tandy did not start life as an electronics retailer. It was founded in 1927 by Norton Hinckley and David Tandy as the Hinckley-Tandy Leather Company, a supplier of leather to shoe repairers in Beaumont, Texas. The first move towards becoming an electronics giant began in 1963 when David Tandy's son, Charles, decided to expand the business, and bought a part share in a floundering Boston-based company called Radio Shack. This company had been operating since the twenties as a small-scale supplier of electrical components to radio hams and other electronics enthusiasts. Although it did most of its business by mail order and had a total of nine shops in the Boston area, it was making large losses. By 1967, Charles Tandy had managed to transform the company's losses of $4,000,000 into a profit of $20,000,000.

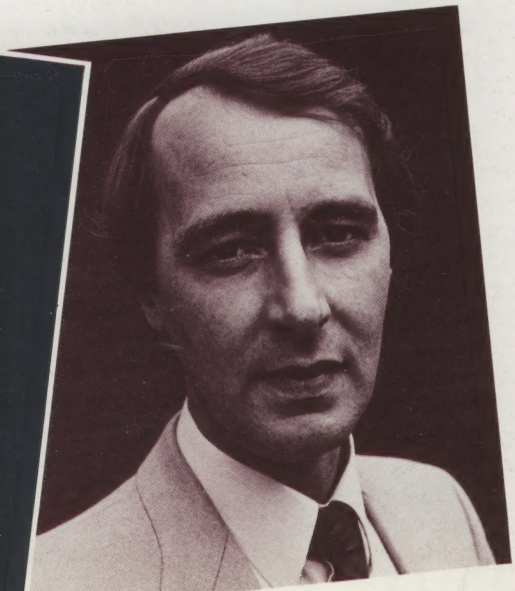
The next significant step was when Tandy took over a chain of department stores called Leonards in 1970, which gave it a foothold in the electrical consumer goods market. This area has now been developed to the point where the 1984 Tandy catalogue features 2,625 non-computer items, ranging from resistors to hi-fi equipment and synthesisers, and 396 computer products.

Tandy arrived in the United Kingdom in 1973 and quickly established itself in the high street as a retailer of electrical goods. In 1978, when the TRS-80 Model I microcomputer was launched in this country, Tandy had 120 stores. By 1983 the number had increased to 227 retail outlets throughout the country. Twenty-six of these are computer centres, specialising solely in computers, software and peripherals.

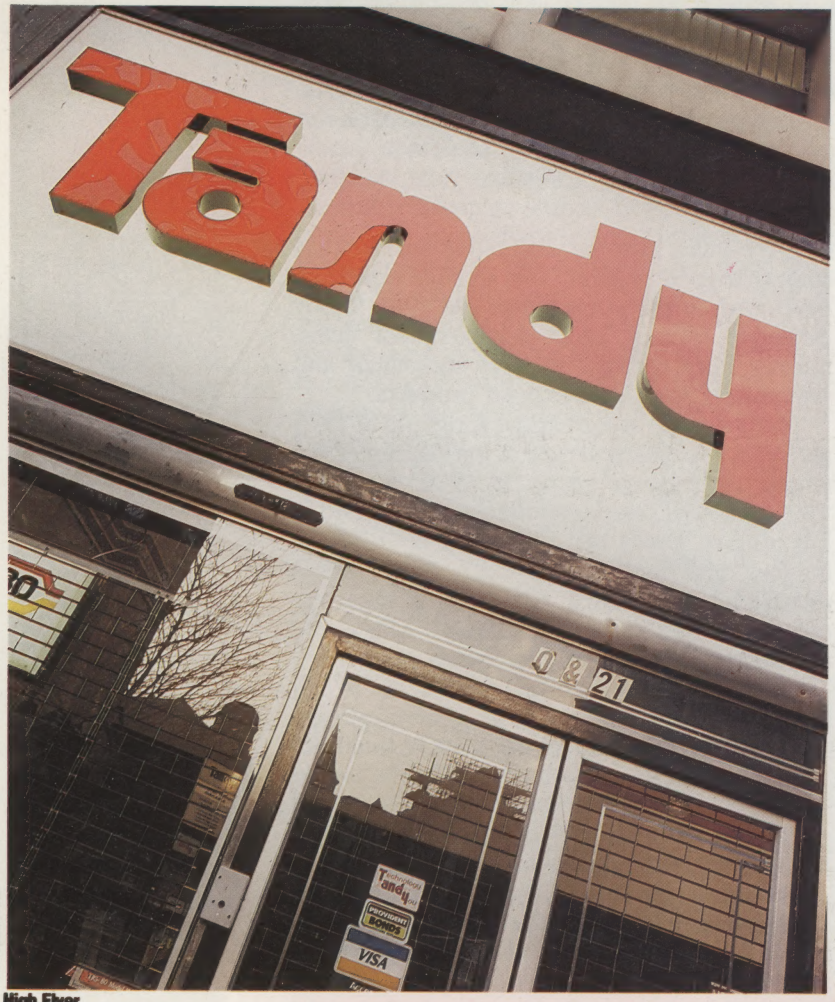
The Model I established Tandy as a top computer manufacturer. It is a single-board machine with a Z80 microprocessor, at least four Kbytes of RAM, and a black-and-white screen with low resolution graphics. Disk drives are available and users can even run the CP/M business operating system on the machine.

Tandy's line has been kept up-to-date ever since, although it has never recaptured the market dominance of the Model I. The company quickly moved into the business micro market with its Model II and currently its range includes the Model 12 and Model 16, both of which are 16-bit machines, as well as a new IBM PC - compatible called the Model 2000. Model I computing moved upmarket with the Model III, which can be regarded either as an expensive home computer or as an economy business machine. Tandy has replaced the III with the Model 4 (and its portable version, the model 4P). These have better business facilities, but retain compatibility

John Sayers, UK Marketing Director



Vince Moore, UK Sales Director



IAN McKINNELL

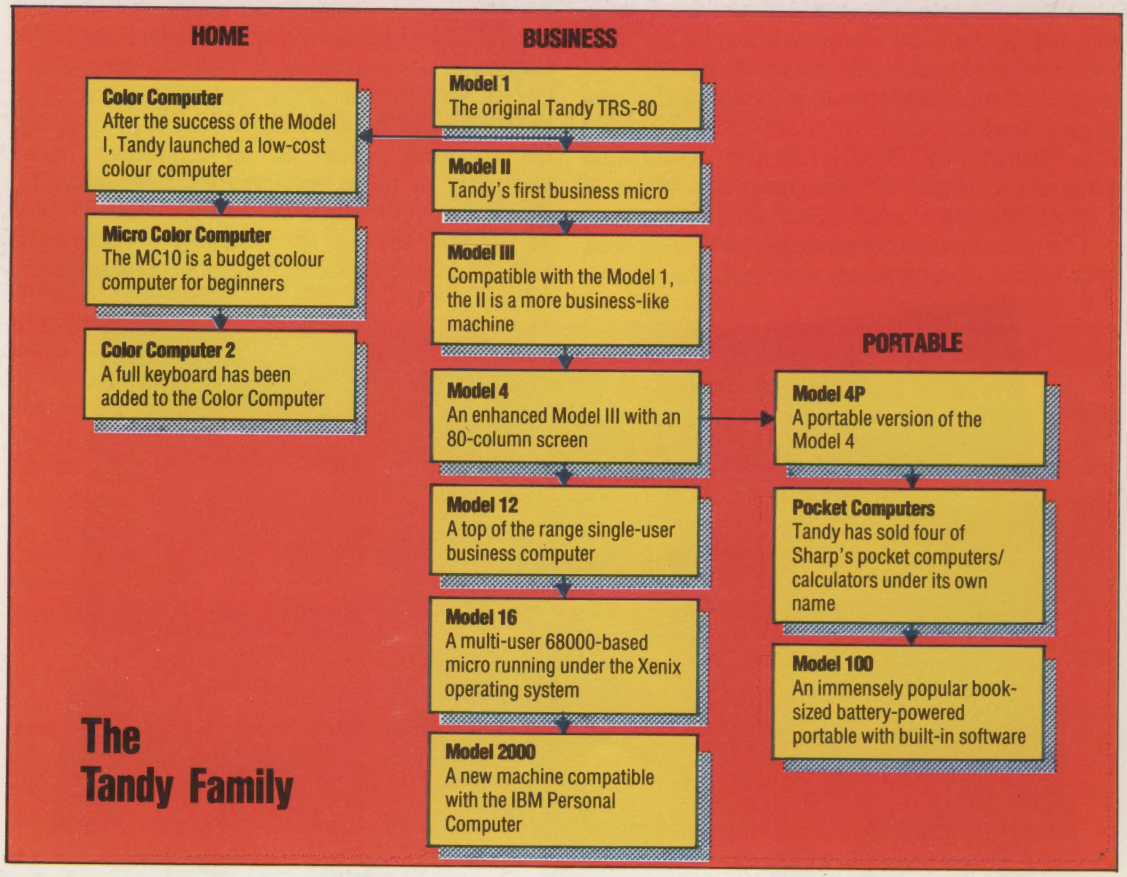
High Flyer
The Tandy high street shop underlines the company's strongest point: a single local supplier for equipment, service and advice. The company's Radio Shack name is equally well known in America

with Model I and III programs — an unusual achievement in microcomputing. As a result, the TRS-80 models support an impressively wide range of software.

Tandy attempted to re-establish itself in the home market with the Tandy Color Computer, a 6809-based micro that shares many of the same features as the Dragon. Although never very successful in the UK, the 'CoCo' has sold well in the USA. A recent update and a mini-version called the Micro Color Computer indicate that Tandy has not given up in this area.

Perhaps the most interesting line has been portable computers. Tandy became involved by selling a series of pocket computers based on the Sharp range. More recently, a deal with another Japanese company, Kyocera, and the American Microsoft software house produced the Model 100, a book-sized battery-powered portable, complete with built-in BASIC, word processor, communications software and a diary.

By 1984, several major firms in the microcomputer industry were beginning to suffer large losses, but business is looking healthy for Tandy, with a steady stream of new products from its design centres in Fort Worth, Texas, where the company is now based, and from the subsidiary TC Electronics Corporation based in Tokyo. With the advantages of a large manufacturing division and its wide network of retailers, the Tandy Corporation seems to be uniquely placed to take advantage of whatever developments may occur in the microelectronics business, and promises to be a long-term survivor.


The Tandy Family
HOME
Color Computer
After the success of the Model I, Tandy launched a low-cost colour computer
Micro Color Computer
The MC10 is a budget colour computer for beginners
Color Computer 2
A full keyboard has been added to the Color Computer
BUSINESS
Model 1
The original Tandy TRS-80
Model II
Tandy's first business micro
Model III
Compatible with the Model 1, the II is a more business-like machine
Model 4
An enhanced Model III with an 80-column screen
Model 12
A top of the range single-user business computer
Model 16
A multi-user 68000-based micro running under the Xenix operating system
Model 2000
A new machine compatible with the IBM Personal Computer
PORTABLE
Model 4P
A portable version of the Model 4
Pocket Computers
Tandy has sold four of Sharp's pocket computers/ calculators under its own name
Model 100
An immensely popular book-sized battery-powered portable with built-in software

GET MACHINE CODE TAPED WITH

CHAMP

* CHAMP: A uniquely comprehensive aid to machine code programming

* Specially commissioned to accompany the *Home Computer Advanced Course* series on machine code

* Suitable for the BBC Model B, Commodore 64 and 48K Sinclair Spectrum

* Pre-recorded on tape cassette FREE with Issue 11 of *The Home Computer Advanced Course*

To unlock the full potential of your computer you have to talk to it in its own language — machine code. The *Home Computer Advanced Course* is teaching you to do just that in its unrivalled series on machine code programming. And now you can bring the computer itself to your aid. With the pre-recorded software that is being given away with Issue 11, you can tackle ambitious programs — thanks to these features, rarely found together in one package:

The Editor enables you to enter your programs, written in Assembly language mnemonics, from the keyboard, or to load them from tape. You can then modify the program freely, with on-entry syntax checking.

The Assembler translates your Assembly language program into machine code — saving you hours of tricky, error-prone work

The Monitor displays contents of memory, together with hex and ASCII equivalents, and enables you to move blocks of memory, search for specified strings, and modify the memory contents directly

The Disassembler translates the machine code version of the program back into Assembly language mnemonics for your scrutiny

The Debug Facility enables you to run the program step by step while displaying the contents of the microprocessor's registers and modifying them as necessary

Normally you'd have to buy several different software packages to obtain all these facilities — and you'd have to spend pounds. They're *free* with Issue 11 of *The Home Computer Advanced Course*. This suite of advanced programs was specially commissioned from a leading software company, Personal Software Services, to be a valuable programming aid for readers. With its aid you'll learn how to make your micro do things that will amaze you: seemingly instantaneous program execution and dazzling, lightning-fast graphics animation.

(CHAMP — Comprehensive Home Computer User's Assembler/ Monitor Package)

GET MACHINE CODE TAPED WITH ISSUE 11 OF THE HOME COMPUTER ADVANCED COURSE

DRAG 64